

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE DEVICE

OBJECT MODULE PLACED IN DEVICE.OBJ

COMPILER INVOKED BY: PLM80 DEVICE,PLM PAGewidth(100) DEBUG OPTIMIZE

```

1      $ TITLE('CP/M 3.0 --- DEVICE')
      device:
      do;

/*
      Copyright (C) 1982
      Digital Research
      P.O. Box 579
      Pacific Grove, CA 93950
*/

/*
      Written: 09 July 82 by John Knight
      Revised  02 Dec 82 by Bruce Skidmore
*/

/*****
*
*      LITERALS AND GLOBAL VARIABLES
*
*****/

2  1  declare
      true      literally '1',
      false     literally '0',
      forever   literally 'while true',
      lit        literally 'literally',
      proc       literally 'procedure',
      dcl        literally 'declare',
      addr       literally 'address',
      cr         literally '13',
      lf         literally '10',
      ctrlc      literally '3',
      ctrlx      literally '18h',
      bksp       literally '8',
      conin$disp literally '22h',
      conout$disp literally '24h',
      auxin$disp literally '26h',
      auxout$disp literally '28h',
      listout$disp literally '2ah',
      mb$input   literally '1',
      mb$output  literally '2',
      mb$in$out  literally '3',
      mb$soft$baud literally '4',
      mb$serial  literally '8',
      mb$xon$xonff literally '16',
      dev$table$adr$func literally '20',
      dev$init$func literally '21',
      cpmversion literally '30h',
      console$page$offset literally '1ch',
      console$width$offset literally '1ah';

```

CP/M Plus Ver 3.0
 COPYRIGHT ©1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950
 SER. # *CP3/135*

```

3 1 declare begin$buffer address;
4 1 declare buf$length byte;
5 1 declare con$width byte;
6 1 declare con$page byte;
7 1 declare phys$dev$table$adr address;
8 1 declare no$chars byte;
9 1 declare string$adr address;
10 1 declare i byte;
11 1 declare device$bit$table (16) byte;
12 1 declare memory (255) byte; /* assignment input buffer */
/* scanner variables and data */
13 1 declare
    options(*) byte data
        ('NAMES'VALUES'HELP'CON:'CONIN:'CONOUT:'LST:''',
         'AUX:'AUXIN:'AUXOUT:'CONSOLE'KEYBOARD'',
         'PRINTER'AUXILIARY'AXI:'AXO:',0ffh),

    options$offset(*) byte data
        (0,6,13,18,23,30,38,43,48,55,63,71,80,88,98,103,107),

    mods(*) byte data
        ('XON'NOXON'NULL'50 75 '110'134'150'300'',
         '600'1200'1800'2400'3600'4800'7200'',
         '9600'19200',0ffh),

    mods$offset(*) byte data
        (0,4,10,15,21,27,31,35,39,43,47,52,57,62,
         67,72,77,82,87),

    page$options (*) byte data
        ('COLUMNS'LINES'PAGESIZE',0ffh),

    page$offsets (*) byte data
        (0,8,14,22),

    end$list byte data (0ffh),

    delimiters(*) byte data (0,'[]=',',',0,0ffh),

    SPACE byte data(5),
    j byte initial(0),
    buf$ptr address,
    index byte,
    endbuf byte,
    delimiter byte;

14 1 declare end$of$string byte initial ('');

/* tables */
15 1 declare phys$table (15) structure
    (name(6) byte,
     characteristic byte,
     baud byte);

16 1 declare biospb structure
    (func byte,

```

COPYRIGHT ©1982
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

        areg byte,
        bcreg address,
        dereg address,
        hlreg address);

17 1      declare scbpd structure
        (offset byte,
         set   byte,
         value address);

18 1      declare baud$rates (*) byte data
        ('NONE 50 75 110 134 150 300 600 ',
         '1200 1800 2400 3600 4800 7200 9600 19200');

19 1      declare baud$table (16) structure
        (graphic (5) byte) at (.baud$rates(0));

20 1      declare log$offsets (*) byte data
        (0,conin$disp,conout$disp,listout$disp,1,auxin$disp,
         auxout$disp,3,conin$disp,listout$disp,2,auxin$disp,auxout$disp);

21 1      declare characteristics$table (*) byte data
        ('INPUT  $OUTPUT  $SOFT-BAUD$SERIAL  $XON-XOFF $');

22 1      declare char$table (5) structure
        (graphic (10) byte) at (.characteristics$table(0));

23 1      declare plm label public;

        /*****
        *
        *      B D O S   INTERFACE
        *
        *****/

24 1      mon1:
        procedure (func,info) external;
25 2          declare func byte;
26 2          declare info address;
27 2      end mon1;

28 1      mon2:
        procedure (func,info) byte external;
29 2          declare func byte;
30 2          declare info address;
31 2      end mon2;

32 1      mon3:
        procedure (func,info) address external;
33 2          declare func byte;
34 2          declare info address;
35 2      end mon3;

36 1      declare cmdrv      byte      external; /* command drive */
37 1      declare fcb (1)    byte      external; /* 1st default fcb */
38 1      declare fcb16 (1)  byte      external; /* 2nd default fcb */

```

COPYRIGHT ©1982
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950
 SER. # _____

```

39 1 declare pass0 address external; /* 1st password ptr */
40 1 declare len0 byte external; /* 1st passwd length */
41 1 declare pass1 address external; /* 2nd password ptr */
42 1 declare len1 byte external; /* 2nd passwd length */
43 1 declare tbuff (1) byte external; /* default dma buffer */

```

```

/*****
*
*      B D O S   Externals
*
*****/

```

```

44 1 printchar:
    procedure(char);
45 2 declare char byte;
46 2 call mon1(2,char);
47 2 end printchar;

48 1 print$buf:
    procedure (buffer$address);
49 2 declare buffer$address address;
50 2 call mon1 (9,buffer$address);
51 2 end print$buf;

```

COPYRIGHT ©1982
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

52 1 read$console$buf:
    procedure (buffer$address,max) byte;
53 2 declare buffer$address address;
54 2 declare new$max based buffer$address address;
55 2 declare max byte;
56 2 new$max = max;
57 2 call mon1(10,buffer$address);
58 2 buffer$address = buffer$address + 1;
59 2 return new$max; /* actually number of chars input */
60 2 end read$console$buf;

```

```

61 1 version: procedure address;
    /* returns current cp/m version # */
62 2 return mon3(12,0);
63 2 end version;

```

```

64 1 getscbbyte: procedure (offset) byte;
65 2 declare offset byte;
66 2 scbpd.offset = offset;
67 2 scbpd.set = 0;
68 2 return mon2(49,.scbpd);
69 2 end getscbbyte;

```

```

70 1 getscbword:
    procedure (offset) address;
71 2 declare offset byte;
72 2 scbpd.offset = offset;
73 2 scbpd.set = 0;
74 2 return mon3(49,.scbpd);
75 2 end getscbword;

```

```

76 1 setscbbyte:

```

```

      procedure (offset,value);
77  2      declare offset byte;
78  2      declare value byte;
79  2      scbpd.offset = offset;
80  2      scbpd.set = OFFH;
81  2      scbpd.value = double(value);
82  2      call mon1(49,.scbpd);
83  2      end setscbbyte;

```

```

84  1      setscbword:
      procedure (offset,value);
85  2      declare offset byte;
86  2      declare value address;
87  2      scbpd.offset = offset;
88  2      scbpd.set = OFEH;
89  2      scbpd.value = value;
90  2      call mon1(49,.scbpd);
91  2      end setscbword;

```

```

92  1      direct$bios:
      procedure (func) address;
93  2      declare func byte;
94  2      biospb.func = func;
95  2      return mon3(50,.biospb);
96  2      end direct$bios;

```

COPYRIGHT ©1982
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

/*****
*
*      SUBROUTINES
*
*****/

```

```

/* *****

```

*** Option scanner ***

```

*****

```

```

97  1      separator: procedure(character) byte;

      /* determines if character is a
         delimiter and which one */
98  2      declare k byte,
         character byte;

99  2      k = 1;
100  2      loop: if delimiters(k) = end$list then return(0);
102  2      if delimiters(k) = character then return(k); /* null = 25 */
104  2      k = k + 1;
105  2      go to loop;

106  2      end separator;

```

```

107 1  opt$scanner:  procedure(list$ptr,off$ptr,idx$ptr);
                        /* scans the list pointed at by idx$ptr
                        for any strings that are in the
                        list pointed at by list$ptr.
                        Off$ptr points at an array that
                        contains the indices for the known
                        list. Idx$ptr points at the index
                        into the list. If the input string
                        is unrecognizable then the index is
                        0, otherwise > 0.

```

First, find the string in the known list that starts with the same first character. Compare up until the next delimiter on the input. if every input character matches then check for uniqueness. Otherwise try to find another known string that has its first character match, and repeat. If none can be found then return invalid.

To test for uniqueness, start at the next string in the known list and try to get another match with the input. If there is a match then return invalid.

else move pointer past delimiter and return.

P.Balma */

```

108 2  declare
        buff      based buff$ptr (1) byte,
        idx$ptr   address,
        off$ptr   address,
        list$ptr  address;

```

```

109 2  declare
        i          byte,
        j          byte,
        list       based list$ptr (1) byte,
        offsets    based off$ptr (1) byte,
        wrd$pos    byte,
        character  byte,
        letter$in$word byte,
        found$first byte,
        start      byte,
        index      based idx$ptr byte,
        save$index byte,
        (len$new,len$found) byte,
        valid      byte;

```

COPYRIGHT ©1982
DIGITAL RESEARCH
P.O. BOX 579
PACIFIC GROVE, CA 93950

SER. # _____

```

/*****
/*      internal subroutines      */
*****/

```

```

110 2    check$in$list: procedure;
        /* find known string that has a match with
           input on the first character. Set index
           = invalid if none found. */

111 3        declare i    byte;

112 3        i = start;
113 3        wrd$pos = offsets(i);
114 3        do while list(wrd$pos) <> end$list;
115 4            i = i + 1;
116 4            index = i;
117 4            if list(wrd$pos) = character then return;
119 4            wrd$pos = offsets(i);
120 4        end;
        /* could not find character */

121 3        index = 0;
122 3        return;
123 3    end check$in$list;

124 2    setup: procedure;
125 3        character = buff(0);
126 3        call check$in$list;
127 3        letter$in$word = wrd$pos;
        /* even though no match may have occurred, position
           to next input character. */

128 3        i = 1;
129 3        character = buff(1);
130 3    end setup;

131 2    test$letter: procedure;
        /* test each letter in input and known string */

132 3        letter$in$word = letter$in$word + 1;

        /* too many chars input? 0 means
           past end of known string */
133 3        if list(letter$in$word) = end$of$string then valid = false;
        else
135 3        if list(letter$in$word) <> character then valid = false;

        i = i + 1;
138 3        character = buff(i);

139 3    end test$letter;

140 2    skip: procedure;
        /* scan past the offending string;
           position buf$ptr to next string...
           skip entire offending string;
           ie., falseopt=mod, [note: comma or
           space is considered to be group
           delimiter] */

141 3        character = buff(i);
142 3        delimiter = separator(character);
        /* No skip for DEVICE */
143 3        do while ((delimiter < 1) or (delimiter > 6));

```

COPYRIGHT ©1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

```

144 4      i = i + 1;
145 4      character = buff(i);
146 4      delimiter = separator(character);
147 4      end;
148 3      endbuf = i;
149 3      buf$ptr = buf$ptr + endbuf + 1;
150 3      return;
151 3  end skip;

152 2  eat$blanks: procedure;

153 3      declare charac based buf$ptr byte;

```

COPYRIGHT ©1982
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

154 3      do while ((delimiter := separator(charac)) = SPACE);
155 4          buf$ptr = buf$ptr + 1;
156 4      end;

157 3  end eat$blanks;

```

```

/*****
/*      end of internals      */
*****/

```

```

/* start of procedure */

158 2      call eat$blanks;
159 2      start = 0;
160 2      call setup;

```

/* match each character with the option
 for as many chars as input
 Please note that due to the array
 indices being relative to 0 and the
 use of index both as a validity flag
 and as a index into the option/mods
 list, index is forced to be +1 as an
 index into array and 0 as a flag*/

```

161 2      do while index < 0;
162 3          start = index;
163 3          delimiter = separator(character);

```

/* check up to input delimiter */

```

164 3      valid = true;      /* test$letter resets this */
165 3      do while delimiter = 0;
166 4          call test$letter;
167 4          if not valid then go to exit1;
169 4          delimiter = separator(character);
170 4      end;

171 3      go to good;

```

/* input ^= this known string;
 get next known string that
 matches */


```

172 3      exit1:      call setup;
173 3          end;

                        /* fell through from above, did
                        not find a good match*/

174 2      endbuf = i;      /* skip over string & return*/
175 2      call skip;
176 2      return;

                        /* is it a unique match in options
                        list? */

177 2      good:      endbuf = i;
178 2          len$found = endbuf;
179 2          save$index = index;
180 2          valid = false;
181 2      next$opt:
                start = index;
182 2          call setup;
183 2          if index = 0 then go to finished;

                        /* look at other options and check
                        uniqueness */

185 2          len$new = offsets(index + 1) - offsets(index) - 1;
186 2          if len$new = len$found then do;
188 3              valid = true;
189 3              do j = 1 to len$found;
190 4                  call test$letter;
191 4                  if not valid then go to next$opt;
193 4              end;
194 3          end;
195 2          else go to nextopt;

                        /* fell through...found another valid
                        match --> ambiguous reference */

196 2          index = 0;
197 2          call skip;      /* skip input field to next delimiter*/
198 2          return;

199 2      finished:      /* unambiguous reference */
                index = save$index;
200 2          buf$ptr = buf$ptr + endbuf;
201 2          call eat$blanks;
202 2          if delimiter <> 0 then
203 2              buf$ptr = buf$ptr + 1;
                else
204 2              delimiter = 5;
205 2          return;

206 2      end opt$scanner;

/* ----- */

207 1      ucase: procedure (char) byte;
208 2          declare char byte;
209 2          if char >= 'a' then
210 2              if char < '{' then
211 2                  return (char-20h);
212 2          return char;

```

COPYRIGHT ©1932
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

213 2      end ucase;

      /* ----- */

214 1      crlf:  proc;
215 2          call printchar(cr);
216 2          call printchar(lf);
217 2          end crlf;

      /* ----- */

      /* fill string @ s for c bytes with f */
218 1      fill:  proc(s,f,c);
219 2          dcl s addr,
              (f,c) byte,
              a based s byte;

220 2          do while (c:=c-1)<>255;
221 3              a = f;
222 3              s = s+1;
223 3              end;
224 2          end fill;

      /* ----- */

      /* The error processor. This routine prints the command line
         with a caret '^' under the offending delimiter, or sub-string.
         The code passed to the routine determines the error message
         to be printed beneath the command string.          */

225 1      errors: procedure (code);
226 2          declare (code,i,j,nlines,rem) byte;
227 2          declare (string$ptr,tstring$ptr) address;
228 2          declare chr1 based string$ptr byte;
229 2          declare chr2 based tstring$ptr byte;
230 2          declare caret$flag byte;

231 2          print$command: procedure (size);
232 3              declare size byte;
233 3              do j=1 to size; /* print command string */
234 4                  call printchar(chr1);
235 4                  string$ptr = string$ptr + 1;
236 4              end;
237 3              call crlf;
238 3              do j=1 to size; /* print caret if applicable */
239 4                  if .chr2 = buf$ptr then do;
240 5                      caret$flag = true;
241 5                      call printchar('^');
242 5                  end;
243 5                  else
244 4                      call printchar(' ');
245 4                      tstring$ptr = tstring$ptr + 1;
246 4                  end;
247 3              call crlf;
248 3              end print$command;

249 2          caret$flag = false;

```

COPYRIGHT © 1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

250 2      string$ptr,tstring$ptr = begin$buffer;
251 2      if con$width < 40 then con$width = 40; /* minimum size screen assumed */
253 2      nlines = buf$length / con$width; /* determine number lines to print */
254 2      rem = buf$length mod con$width; /* number of extra characters */
255 2      if (code = 2) or (code = 1) then /* adjust carot pointer */
256 2          buf$ptr = buf$ptr - 1; /* for delimiter errors */
      else
257 2          buf$ptr = buf$ptr - endbuf - 1; /* for sub-string errors */
258 2      call crlf;
259 2      do i=1 to nlines;
260 3          tstring$ptr = string$ptr;
261 3          call print$command(con$width);
262 3      end;
263 2      call print$command(rem);
264 2      if carot$flag then
265 2          call print$buf(('.Error at the '^'; $'));
      else
266 2          call print$buf(('.Error at end of line; $'));
267 2      if con$width < 63 then
268 2          call crlf;
269 2      do case code; /* error messages */
270 3          call print$buf(('.Invalid number$'));
271 3          call print$buf(('.End of line expected$'));
272 3          call print$buf(('.Invalid delimiter$'));
273 3          call print$buf(('.Invalid option$'));
274 3          call print$buf(('.Baud rate can not be set for this device$'));
275 3          call print$buf(('.Invalid physical device$'));
276 3          call print$buf(('.Physical device does not have input capability$'));
277 3          call print$buf(('.Physical device does not have output capability$'));
278 3          call print$buf(('.Physical device does not have input/output capability$'));
279 3          call print$buf(('.A NULL device can not be assigned to CONIN$'));
280 3          call print$buf(('.Ambiguous assignments to a NULL device are not allowed$'));
281 3      end;
282 2      call crlf;
283 2      call moni(0,0);
284 2  end errors;

```

```
/* - - - - - */
```

```

/* Help display. A simple print of the syntax accepted by this
utility. The display assumes a minimum 40 column screen and
does not give an explanation to the commands. For quick ref. only

```

```
help: procedure;
```

```

COMMENTED OUT -- NEW HELP
PROGRAM WILL REPLACE THIS
DISPLAY

```

```

call print$buf(
'COMMAND SYNTAX:',cr,lf,cr,lf,
'DEVICE',cr,lf,
'DEVICE NAMES',cr,lf,
'DEVICE VALUES',cr,lf,
'DEVICE pd',cr,lf,
'DEVICE ld',cr,lf,
'DEVICE ld=pd[opt,opt],pd[opt],...',cr,lf,
'DEVICE pd[opt,opt]',cr,lf,
'DEVICE ld=NULL',cr,lf,

```

COPYRIGHT © 1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

'DEVICE CONSOLE[COLUMNS=nnn,LINES=nnn]',cr,lf,
'DEVICE CONSOLE[PAGESIZE]',cr,lf,cr,lf,
'pd = a physical device',cr,lf,
'ld = a logical device',cr,lf,
'    CON!,CONIN!,CONOUT!,LST!,AUX!:',cr,lf,
'    AUXIN!,AXI!,AUXOUT!,AXO!,CONSOLE:',cr,lf,
'    KEYBOARD,PRINTER, or AUXILIARY',cr,lf,
'opt = a valid option',cr,lf,
'    XON,NOXON, or a baud rate: 50,',cr,lf,
'    75,110,134,150,300,600,1200,1800,',cr,lf,
'    2400,3600,4800,7200,9600,19200',cr,lf,
'nnn = a number; 0-255',cr,lf,'$'));
end help; */

```

```
/* ----- */
```

```

285 1  set$bit:
      procedure (val,bit) address;
      /* sets a bit in 0-15 in val, returns val */
286 2  declare bit byte;
287 2  declare val address;
288 2  declare temp address;
289 2  temp = 1;
290 2  bit = 15 - bit;
291 2  if bit < 0 then
292 2      temp = shl(temp,bit);
293 2  val = val or temp;
294 2  return val;
295 2  end set$bit;

```

COPYRIGHT © 1982
DIGITAL RESEARCH
P.O. BOX 579
PACIFIC GROVE, CA 93950

SER. # _____

```
/* ----- */
```

```
/* This routine assigns to a word in the system control block a
   bit pattern as specified in the device$bit$table.      */
```

```

296 1  make$assignments: procedure (offset);
297 2  declare (i,offset) byte;
298 2  declare val address;
299 2  val = 0; /* clear address to be set */
300 2  do i=0 to 15;
301 3      if device$bit$table(i) = 1
          then val= set$bit(val,i);
303 3  end;
304 2  call set$cbword(offset,val);
305 2  end make$assignments;

```

```
/* ----- */
```

```
/* This routine prints the physical device located in the
   physical device table at the index passed to the routine */
```

```

306 1  print$phys$device: procedure (index);
307 2  declare (i,index) byte;
308 2  do i=0 to 5;
309 3      call printchar(phys$table(index).name(i));
310 3  end;
311 2  end print$phys$device;

```

```

/* ----- */

/* This routine prints the baud rate corresponding to the baud
   code found in the physical device table. The index to the
   physical device table is passed to this routine. */

312 1  print$baud$rate; procedure (index);
313 2      declare (k,index,baud) byte;
314 2      baud = phys$table(index).baud;
315 2      if baud > 15 then baud = 0;
317 2      do k=0 to 4;
318 3          call printchar(baud$table(baud).graphic(k));
319 3      end;
320 2  end print$baud$rate;

/* ----- */

/* This routine prints the physical characteristics codes for
   a specific physical device found in the physical device table.
   This procedure is called by names. */

321 1  print$phys$characteristics; procedure (index);
322 2      declare (char,index,ct) byte;
323 2      ct = 0;
324 2      char = phys$table(index).characteristic;
325 2      char = shr(char,1);
326 2      if carry = 0ffh then do; /* input bit */
328 3          call printchar('I');
329 3          ct = ct + 1;
330 3      end;
331 2      char = shr(char,1);
332 2      if carry = 0ffh then do; /* output bit */
334 3          call printchar('O');
335 3          ct = ct + 1;
336 3      end;
337 2      char = shr(char,2); /* skip soft-baud */
338 2      if carry = 0ffh then do; /* serial bit in carry */
340 3          call printchar('S');
341 3          ct = ct + 1;
342 3      end;
343 2      char = shr(char,1);
344 2      if carry = 0ffh then do; /* xon-xoff bit */
346 3          call printchar('X');
347 3          ct = ct + 1;
348 3      end;
349 2      do while ct <> 4;
350 3          call printchar(' ');
351 3          ct = ct + 1;
352 3      end;
353 2  end print$phys$characteristics;

/* ----- */

/* This routine prints the names of the physical devices as well
   as the baud rate and characteristics codes. */

```

COPYRIGHT ©1982

DIGITAL RESEARCH

P.O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

```

354 1  names: procedure;
355 2      declare (i,j,cols,char,baud,k) byte;
356 2      call crlf;
357 2      call print$buf(.(('Physical Devices: ',cr,lf,'$')));
358 2      call print$buf(.(('I=Input,O=Output,S=Serial,X=Xon-Xoff',cr,lf,'$')));
359 2      i = con$width;
360 2      if i < 40 then i = 40;
362 2      cols = i / 20; /* determine columns per line */
363 2      j = 0; /* table index */
364 2      crloop: i=i; /* columns counter */
365 2      process: if phys$table(j).name(0) = 0 then do;
367 3          call crlf;
368 3          return;
369 3      end;

/* print device name, baud, and attributes */
370 2      call print$phys$device(j);
371 2      call printchar(' ');
372 2      call print$baud$rate(j);
373 2      call printchar(' ');
374 2      call print$phys$characteristics(j);
375 2      call print$buf(.((' $')));
376 2      j = j + 1;
377 2      if i >= cols then do;
379 3          call crlf;
380 3          goto crloop;
381 3      end;
382 2      i = i + 1;
383 2      goto process;
384 2  end names;

```

COPYRIGHT ©1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

/* ----- */

/* This routine prints the physical devices that are assigned
 to the logical device. The bit pattern of the vector passed
 to this routine determines the current assignments to the device */

```

385 1  show$physical$devices:
      procedure (vector);
386 2      declare vector address;
387 2      declare device$present byte;
388 2      declare bit$table (16) byte;
389 2      declare (i,k,cols,max) byte;
390 2      i = con$width;
391 2      if i < 40 then i = 40;
393 2      cols = (i - 10) / 7; /* determine phys$devices per line */
394 2      do i = 0 to 15;
395 3          vector = shl(vector,1);
396 3          bit$table(i) = carry; /* ff = 1, 0 = 0 */
397 3      end;
398 2      i = 0;
399 2      do while phys$table(i).name(0) <> 0;
400 3          if i = 15 then goto set$max;
402 3          i = i + 1;
403 3      end;
404 2      set$max: max = i; /* number of entries in table */
405 2      device$present = false;
406 2      k = 1; /* cols printed count */

```

```

407 2      do i = 0 to 14;
408 3          if bit$table(i) = Offh then do;
/* obtain match from physical device table */
410 4              if i > max then do;
412 5                  call print$buf(,('Bad Logical Device Assignment; $'));
413 5                  call print$buf(,('Physical Device Does Not Exist$'));
414 5                  call crlf;
415 5                  return;
416 5              end;
417 4              device$present = true;
418 4              call print$phys$device(i);
419 4              call printchar(' ');
420 4              k = k + 1;
421 4              if k > cols then do;
423 5                  k = 1;
424 5                  call crlf;
425 5                  call print$buf(,('          $'));
426 5                  end;
427 4              end;
428 3          end;
429 2          if bit$table(15) = Offh then do; /* File assignment */
431 3              device$present = true;
432 3              call print$buf(,('File$'));
433 3              end;
434 2          if (device$present = false) then
435 2              call print$buf(,('Null Device$'));
436 2              call crlf;
437 2          end show$physical$devices;

```

/* ----- */

/* This procedure produces the values display. It shows all the assignments of physical devices to the logical devices */

```

438 1  values: procedure;
439 2      declare val address;
440 2      call crlf;
441 2      call print$buf(,('Current Assignments: ',crlf,'$'));
442 2      val = getscbword(conin$disp);
443 2      call print$buf(,('CONIN: = $'));
444 2      call show$physical$devices(val);
445 2      val = getscbword(conout$disp);
446 2      call print$buf(,('CONOUT: = $'));
447 2      call show$physical$devices(val);
448 2      val = getscbword(auxin$disp);
449 2      call print$buf(,('AUXIN: = $'));
450 2      call show$physical$devices(val);
451 2      val = getscbword(auxout$disp);
452 2      call print$buf(,('AUXOUT: = $'));
453 2      call show$physical$devices(val);
454 2      val = getscbword(listout$disp);
455 2      call print$buf(,('LST: = $'));
456 2      call show$physical$devices(val);
457 2      call crlf;
458 2      end values;

```

/* ----- */

COPYRIGHT © 1982
DIGITAL RESEARCH
P.O. BOX 579
PACIFIC GROVE, CA 93950

SER. # _____

```

/* This procedure searches for the string pointed to by
   search$string$adr in the local physical device table.
   The length of the input string is determined by endbuf. */

```

```

459 1  search$physical$table;
      procedure (search$string$adr) byte;
460 2      declare (i,j) byte;
461 2      declare search$string$adr address;
462 2      declare string (6) byte;
463 2      declare loc based search$string$adr (6) byte;
464 2      if endbuf > 6 then return Offh;
466 2      call fill(.string(0),' ',6);
467 2      do i=0 to (endbuf-1);
468 3          string(i)=loc(i);
469 3      end;
470 2      i = 0;
471 2      do while phys$table(i).name(0) <> 0;
472 3          do j=0 to 5;
473 4              if string(j) <> phys$table(i).name(j)
                  then goto search$next;
475 4          end;
476 3          return i; /* found; return index */
477 3          search$next: i=i+1;
478 3          if i > 15 then return Offh; /* not found */
480 3      end;
481 2      return Offh; /* not found, table empty */
482 2  end search$physical$table;

```

COPYRIGHT ©1982
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

/* ----- */

```

```

/* This routine processes the physical device options: 'XON','NOXON'
   and the baud rates. It calls the scanner and processes on the fly */

```

```

483 1  process$option: procedure (table$index);
484 2      declare table$index byte;
485 2      declare soft$baud byte;
486 2      declare (char,baud) byte;
487 2      declare val address;
488 2      char = phys$table(table$index).characteristic;
489 2      baud = phys$table(table$index).baud;
490 2      index = 0;
491 2      delimiter = 1;
492 2      do while((delimiter <> 2) and (delimiter <> 6));
493 3          call opt$scanner(.mods(0),.mods$offset(0),.index);
494 3          if index = 0 then call errors(3);
496 3          if index = 3 then call errors(3);
498 3          if index = 1 then /* Xon */
499 3              phys$table(table$index).characteristic = char or ab$xon$xoff;
500 3          if index = 2 then /* No Xon */
501 3              phys$table(table$index).characteristic = char and (not ab$xon$xoff);
502 3          if index > 2 then do; /* baud rates to be set */
504 4              index = index - 3;
                  /* set baud rate only if soft$baud set to 1 */
505 4              soft$baud = shr(char,3);
506 4              soft$baud = carry; /* Offh = 1, 0 = 0 */
507 4              if soft$baud = 0 then

```



```

508 4      call errors(4);
          /* set baud in table and have bios initialize device */
509 4      phys$table(table$index).baud = index;
          /* move local phys$device table to actual table in bios */
510 4      call move(120,,phys$table(0),phys$dev$table$adr);
511 4      biospb,bcreg = double(table$index);
512 4      val = direct$bios(dev$init$func);
513 4      end;
          else
514 3      call move(120,,phys$table(0),phys$dev$table$adr);
515 3      end;
516 2      end process$option;

```

```
/* ----- */
```

```
/* This routine converts an ascii number string into a byte number,
   ie. 32h 35h 35h ==> FFh in one byte. Numbers allowed are 0-255 */
```

```

517 1      number: procedure (loc,length) byte;
518 2      declare (loc,val) address;
519 2      declare (length,i) byte;
520 2      declare chr based loc byte;
521 2      if length > 3 then
522 2      call errors(0);
523 2      val = 0;
524 2      do i=1 to length;
525 3      if (chr < 30h) or (chr > 39h) then
526 3      call errors(0);
527 3      val = val * 10 + (chr - 30h);
528 3      loc = loc + 1;
529 3      end;
530 2      if val > 255 then
531 2      call errors(0);
532 2      return low(val);
533 2      end number;

```

COPYRIGHT ©1982
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```
/* ----- */
```

```
/* This routine converts a byte into an ascii string of numbers,
   printing the number to the screen. ie. FFh ==> 255 */
```

```

534 1      print$byte: procedure (num);
535 2      declare (hundreds,tens,ones,num) byte;
536 2      hundreds = num / 100;
537 2      num = num - (100 * hundreds);
538 2      tens = num / 10;
539 2      ones = num - (10 * tens);
540 2      if hundreds > 0 then
541 2      call printchar(hundreds + 30h);
542 2      if (hundreds > 0) or (tens > 0) then
543 2      call printchar(tens + 30h);
544 2      call printchar(ones + 30h);
545 2      end print$byte;

```

```
/* ----- */
```

```
/* This procedure processes the console page setting options.
```

It parses the command options and sets the scb page accordingly.
The result of the process is displayed showing the user the
number of lines and columns of the console. */

```

546 1  process$page$options: procedure;
547 2      declare num byte;
548 2      delimiter=1;
549 2      index=0;
550 2      do while ((delimiter < 2) and (delimiter < 6));
551 3          call opt$scanner(.page$options(0),.page$offsets(0),.index);
552 3          if index = 0 then /* bad option */
553 3              call errors(3);
554 3          if index = 1 then do; /* columns */
555 4              if delimiter < 3 then /* '=' */
556 4                  call errors(2);
557 4              else do;
558 4                  call opt$scanner(.page$options(0),.page$offsets(0),.index);
559 5                  num = number(buf$ptr-endbuf-1,endbuf)-1;
560 5                  call setscbbyte(console$width$offset,num);
561 5              end;
562 5          end;
563 4          end;
564 3          if index = 2 then do; /* lines */
565 4              if delimiter < 3 then
566 4                  call errors(2);
567 4              else do;
568 4                  call opt$scanner(.page$options(0),.page$offsets(0),.index);
569 5                  num = number(buf$ptr-endbuf-1,endbuf)-1;
570 5                  call setscbbyte(console$page$offset,num);
571 5              end;
572 5          end;
573 4          end;
574 3          end;
575 2          con$width = getscbbyte(console$width$offset);
576 2          con$page = getscbbyte(console$page$offset);
577 2          call crlf;
578 2          call print$buf(.(('Console width set to $')));
579 2          call print$byte(con$width+1);
580 2          call print$buf(.((' columns',crlf,'Console page set to $')));
581 2          call print$byte(con$page+1);
582 2          call print$buf(.((' lines',crlf,'$')));
583 2          call crlf;
584 2          call monl(0,0);
585 2      end process$page$options;

```

/* ----- */

/* This routine produces the display of the assignments to an
individual logical device. The command that invokes this
procedure is 'DEVICE <logical device>'. */

```

586 1  show$assignments: procedure (index);
587 2      declare (index,offset) byte;
588 2      declare val address;
589 2      offset = log$offsets(index-4);
590 2      if (offset = 0) or (offset = 3) then do; /* CON: */
591 3          call print$buf(.(('CONIN: = $')));
592 3          val = getscbword(conin$disp);
593 3          call show$physical$devices(val);
594 3      end;

```

COPYRIGHT © 1982

DIGITAL RESEARCH

P.O. BOX 579

PACIFIC GROVE, CA 93950

SER. #

```

595 3      call print$buf(.(('CONOUT: = $')));
596 3      val = getscbword(conout$disp);
597 3      call show$physical$devices(val);
598 3      end;
599 2      if (offset = 1) or (offset = 2) then do; /* AUX: */
601 3          call print$buf(.(('AUXIN: = $')));
602 3          val = getscbword(auxin$disp);
603 3          call show$physical$devices(val);
604 3          call print$buf(.(('AUXOUT: = $')));
605 3          val = getscbword(auxout$disp);
606 3          call show$physical$devices(val);
607 3      end;
608 2      if offset > 3 then do; /* all others */
610 3          do case (offset - 22h);
611 4              call print$buf(.(('CONIN: = $')));
612 4              ;
613 4              call print$buf(.(('CONOUT: = $')));
614 4              ;
615 4              call print$buf(.(('AUXIN: = $')));
616 4              ;
617 4              call print$buf(.(('AUXOUT: = $')));
618 4              ;
619 4              call print$buf(.(('LST: = $')));
620 4          end;
621 3          val = getscbword(offset);
622 3          call show$physical$devices(val);
623 3      end;
624 2      call crlf;
625 2      call mon1(0,0);
626 2      end show$assignments;

```

COPYRIGHT ©1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

/* - - - - - */

/* This routine is called if the first sub-string in the command
 line was determined to be a logical device. If an end-of-line
 is the delimiter, the routine will display the assignments to
 the specified logical device. If a '[' is found as the delimiter
 and the logical device is console, then the option processor to
 set the console page parameters is called. If the delimiter was
 an '=' then an assignment of physical devices to the logical
 device is done. */

```

627 1      found$logical$device: procedure;
628 2          declare (save$index,offset,eoln,i,val) byte;
629 2          declare next$delim based buf$ptr byte;
630 2          save$index = index; /* save index to logical device */
631 2          if (delimiter = 0) or (delimiter = 6)
              then call show$assignments(index); /* DEVICE <log. dev> */
              else do;
633 2              if delimiter = 1 then do; /* '[' */
634 3                  if (index=4) or (index=5) or (index=6) or (index=11) then
636 4                      call process$page$options; /* DEVICE CON:[col=45,lines=21] */
637 4                  else
638 4                      call errors(2);
639 4                  end;
640 3              else if delimiter <> 3 then
641 3                  call errors(2);

```

```

end;
643 2 delimiter = 1; /* do assignment: DEVICE CON:=CRT,CRT1[XON,1200],... */
644 2 index = 0;
645 2 call opt$scanner(.mods(0),.mods$offset(0),.index);
646 2 offset = log$offsets(save$index - 4);
647 2 if index = 3 then do; /* NULL */
649 3   if (offset < 4) then do; /* CON: and AUX:*/
651 4     call errors(10);
652 4   end;
653 3   else do;
654 4     if (offset=conin$disp) then do;
656 5       call errors(9);
657 5     end;
658 4     else do;
659 5       call setsbword(offset,0);
660 5     end;
661 4   end;
662 3 end;
663 2 else do; /* Process physical name */
664 3   eoln = false;
665 3   do i = 0 to 15; /* clear bit table */
666 4     device$bit$table(i) = 0;
667 4   end;
668 3   do while not eoln;
669 4     val = search$physical$table(buf$ptr-endbuf-1);
670 4     if val = 0fff then /* not found */
671 4       call errors(5);
672 4     device$bit$table(val) = 1; /* mark bit to be set in log device vector */
673 4     if delimiter = 1 then
674 4       call process$option(val);
675 4     if (delimiter=0) or (delimiter=6) or ((delimiter=2) and (next$delim=0))
676 4       then eoln = true;
677 4     if ((delimiter = 2) and (next$delim = ',')) then
678 4       buf$ptr = buf$ptr + 1; /* case where 2 delimiters: ', ' */
679 4     if not eoln then
680 4       call opt$scanner(.mods(0),.mods$offset(0),.index);
681 4     end;
682 3   if (offset = 0) or (offset = 3) then do; /* CON: */
684 4     if ((phys$table(val).characteristic and mb$in$out)=mb$in$out) then do;
686 5       call make$assignments(conin$disp);
687 5       call make$assignments(conout$disp);
688 5     end;
689 4     else call errors(8);
690 4   end;
691 3   else do;
692 4     if ((offset=1) or (offset=2)) then do; /* AUX: */
694 5     if ((phys$table(val).characteristic and mb$in$out)=mb$in$out) then do;
696 6       call make$assignments(auxin$disp);
697 6       call make$assignments(auxout$disp);
698 6     end;
699 5     else call errors(8);
700 5   end;
701 3   else do;
702 5     if ((offset=conin$disp) or (offset=auxin$disp)) then do;
704 6     if ((phys$table(val).characteristic and mb$input)<> mb$input)
705 6       then call errors(6);
706 6     else call make$assignments(offset); /* CONIN: OR AUXIN: */

```

COPYRIGHT ©1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

707 6      end;
708 5      else do;
709 6          if ((phys$table(val).characteristic and mb$output) <> mb$output)
              then call errors(7);
711 6          else call make$assignments(offset); /* CONOUT; OR AUXOUT; OR LSTOUT; */
712 6      end;
713 5      end;
714 4      end;
715 3      end;
716 2      end found$logical$device,

```

```
/* ----- */
```

```

/* This routine produces the display invoked by the command
   string: 'DEVICE <physical device>'. It prints the characteristics
   of the device as found in the physical device table */

```

```

717 1      show$characteristics: procedure (index);
718 2          declare (index,char,baud,j,i) byte;
719 2          char = phys$table(index).characteristic;
720 2          baud = phys$table(index).baud;
721 2          call crlf;
722 2          call print$buf(,('Physical Device:  '));
723 2          call print$phys$device(index);
724 2          call crlf;
725 2          call print$buf(,('Baud Rate:      '));
726 2          call print$baud$rate(index);
727 2          call crlf;
728 2          call print$buf(,('Characteristics:  '));
729 2          do i=0 to 4;
730 3              char = shr(char,i);
731 3              if carry = 0fff then do;
733 4                  call print$buf(,char$table(i));
734 4                  call crlf;
735 4                  do j=0 to 17;
736 5                      call printchar(' ');
737 5                  end;
738 4              end;
739 3          else do;
740 4              if i = 3 then do;
742 5                  call print$buf(,('PARALLEL'));
743 5                  call crlf;
744 5                  do j=0 to 17;
745 6                      call printchar(' ');
746 6                  end;
747 5              end;
748 4          end;
749 3      end;
750 2      call mon1(0,0);
751 2      end show$characteristics;

```

```
/* ----- */
```

```

/* This routine is called whenever a presumed physical device
   is found as the first entry by the parser. It looks up the
   string in the physical device table to validate it. If the
   device has options, it calls process option to set the baud & protocol */

```

COPYRIGHT © 1982

DIGITAL RESEARCH

P.O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

```

752 1    found$physical$device; procedure (string$adr);
753 2        declare (eoln,index) byte;
754 2        declare string$adr address;
755 2        if (delimiter=0) or (delimiter=6) then
756 2            eoln = true;
            else
757 2            eoln = false;
758 2            index = search$physical$table(string$adr);
759 2            if index = 0fff then
760 2                call errors(5);
761 2            if eoln then      /* DEVICE <phys.dev> */
762 2                call show$characteristics(index);
763 2            if delimiter = 1 then
764 2                call process$option(index);
            else
765 2                call errors(2);
766 2        end found$physical$device;

/* ----- */

/* This routine determines which of the sub-routines should
   continue with the parsing and eventual execution of the
   command string. In the event that the commands were 'NAMES',
   'VALUES', no further parsing is needed and the routines
   are called directly to produce the desired displays.    */

767 1    parser; procedure;
768 2        declare (t,char,i) byte;
769 2        declare eoln byte;
770 2        declare phys$dev byte;
771 2        declare log$dev byte;
772 2        delimiter = 1;
773 2        index = 0;
774 2        if tbuff(0) = 0 then
775 2            begin$buffer,buf$ptr = .memory(2);
776 2        else do;
777 3            buf$ptr = .tbuff(2);
778 3            begin$buffer = .tbuff(1);
779 3            buf$length = tbuff(0);
780 3        end;
781 2        call opt$scanner(.options(0),.options$offset(0),.index);
782 2        if (delimiter=0) or (delimiter=2) or (delimiter=6) then
783 2            eoln = true;
            else
784 2            eoln = false;
785 2            if (index = 0) or (index = 3) then do; /* HELP is now a valid phys device */
787 3                call found$physical$device(buf$ptr-endbuf-1);
788 3                call names;      /* show results */
789 3                call values;
790 3            end;
791 2            else do;
792 3                if index = 1 then do; /* names */
794 4                    if eoln then
795 4                        call names;
                    else
796 4                        call errors(1);

```

COPYRIGHT © 1982
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950
 SER. # _____

```

797 4      end;
798 3      else do;
799 4          if index = 2 then do; /* values */
801 5              if eoln then
802 5                  call values;
803 5              else
804 5                  call errors(1);
805 4          end;
806 5          else do;
807 5              call found$logical$device;
808 5              call names;      /* show results */
809 5              call values;
810 4          end;
811 3      end;
812 2      end parser;

```

COPYRIGHT © 1982
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

/* ----- */

```

813 1      input$found: procedure (buffer$adr) byte;
814 2          declare buffer$adr address;
815 2          declare char based buffer$adr byte;
816 2          do while (char = ' ') or (char = 9); /* tabs & spaces */
817 3              buffer$adr = buffer$adr + 1;
818 3          end;
819 2          if char = 0 then /* eoln */
820 2              return false; /* input not found */
821 2          else
822 2              return true; /* input found */
823 2          end input$found;

```

/* ----- */

```

/*****
*
*      M A I N   P R O G R A M
*
*****/

```

```

823 1      plm:
824 2          do;
825 3              if (low(version) < cpmversion) or (high(version) = 1) then do;
826 3                  call print$buf(,('Requires CP/M 3.0$'));
827 3                  call mon1(0,0);
828 3              end;
829 2          phys$dev$table$adr = direct$bios(dev$table$adr$func);
830 2          if (tbuff(0) <> 0) and (phys$dev$table$adr = 0) then do;
831 3              buf$ptr = ,tbuff(1);
832 3              call opt$scanner(,options(0),,options$offset(0),,index);
833 3              if ((index = 4) or (index = 11)) and (delimiter = 1) then do;
834 4                  call parser;
835 4                  call mon1(0,0);
836 4              end;
837 3          end;
838 2          if (phys$dev$table$adr = 0) then do;

```

```

842 3      call print$buf(,('Device Reassignment Not Supported$'));
843 3      call mon1(0,0);
844 3      end;
845 2      con$width = getscbbyte(console$width$offset);
846 2      con$page = getscbbyte(console$page$offset);
847 2      call move(120,phys$dev$table$adr,phys$table(0));
848 2      if not input$found(.tbuf(1)) then do;
          /* display names & values and prompt for the assignment */
850 3      call names;
851 3      call values;
852 3      call print$buf(,('Enter new assignment or hit RETURN $'));
          /* can not use default dma; not always enough room for input */
853 3      call crlf;
854 3      no$chars = read$console$buf(.memory(0),255);
855 3      call crlf;
856 3      memory(1) = ' '; /* blank out nc field */
857 3      memory(no$chars+2) = 0; /* mark eoln */
858 3      if not input$found(.memory(1)) then /* no input, quit */
859 3      call mon1(0,0);
860 3      do i=1 to no$chars; /* convert input to caps */
861 4          memory(i+1) = ucase(memory(i+1));
862 4      end;
863 3      buf$length = no$chars;
864 3      end;
865 2      call parser;
866 2      call mon1(0,0);
867 2      end;
868 1      end device;

```

COPYRIGHT © 1982

DIGITAL RESEARCH

P.O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

MODULE INFORMATION:

```

CODE AREA SIZE      = 19FBH  6651D
VARIABLE AREA SIZE = 023CH   572D
MAXIMUM STACK SIZE = 0012H   18D
1333 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-80 COMPILATION


```

0000 DEVICE
0000 MCD80A
0004 BDISK      0080 BUFF      0080 BUFFA      0050 CHDRV      0000 CPU
007C CR          006D DOLLA      005C FCB      006C FCB16      005C FCBA
005C IFCB      005C IFCBA      0003 IOBYTE      0053 LENO      0056 LEN1
0006 MAXB      0006 MEMSIZ      0005 MON1      0005 MON2      0005 MON2A
0005 MON3      0000 OFFSET      006E PARMA      0051 PASS0      0054 PASS1
007F RO          007D RR          007D RRECA      007F RREC0      005C SFCB
0080 TRUFF
0000 DEVICE
1E9E MEMORY      1C62 BEGINBUFFER      1C64 BULENGTH      1C65 CONWIDTH
1C66 CONPAGE      1C67 PHYSDEVTABLEADR      1C69 NOCHARS      1C6A STRINGADR
1C6C I          1C6D DEVICEBITTABLE      1C7D MEMORY      0180 OPTIONS
01EC OPTIONSOFFSET      01FD MODS      0255 MODSOFFSET
0268 PAGEOPTIONS      027F PAGEOFFSETS      0283 ENDLIST
0284 DELIMITERS      028C SPACE      1D7C J      1D7D BUFPTR      1D7F INDEX
1D80 ENDBUF      1D81 DELIMITER      1D82 ENDOFSTRING      1D83 PHYSTABLE
1DFB BIOSPB      1E03 SCBPD      028D BAUDRATES      028D BAUDTABLE      02DD LOGOFFSETS
02EA CHARACTERISTICSTABLE      02EA CHARTABLE      06ED PLM      0848 PRINTCHAR
1E07 CHAR      0358 PRINTBUF      1E08 BUFFERADDRESS
0868 READCONSOLEBUF      1E0A BUFFERADDRESS      1E0C MAX
088F VERSION      0898 GETSCBBYTE      1E0D OFFSET      08B0 GETSCBWORD      1E0E OFFSET
08C8 SETSCBBYTE      1E0F OFFSET      1E10 VALUE      08EA SETSCBWORD      1E11 OFFSET
1E12 VALUE      0910 DIRECTBIOS      1E14 FUNC      0923 SEPARATOR      1E15 CHARACTER
1E16 K          092C LOOP      095A OPTSCANNER      1E17 LISTPTR      1E19 OFFPTR
1E18 IDXPTR      1E1D I          1E1E J          1E1F WRDPOS      1E20 CHARACTER
1E21 LETTERINWORD      1E22 FOUNDFIRST      1E23 START      1E24 SAVEINDEX
1E25 LENNEW      1E26 LENFOUND      1E27 VALID      0A80 CHECKINLIST
1E28 I          0AD6 SETUP      0AF4 TESTLETTER      0B34 SKIP      0B90 EATBLANKS
09BA EXIT1      09CA GOOD      09DF NEXTOPT      0A50 FINISHED      0BAC UCASE
1E29 CHAR      0BCA CRLF      0BD5 FILL      1E2A S          1E2C F
1E2D C          0C00 ERRORS      1E2E CODE      1E2F I          1E30 J
1E31 NLines      1E32 REM      1E33 STRINGPTR      1E35 TSTRINGPTR      1E37 CAROTFLAG
0D53 PRINTCOMMAND      1E38 SIZE      0DBE SETBIT      1E39 VAL
1E3B BIT      1E3C TEMP      0DF7 MAKEASSIGNMENTS      1E3E OFFSET
1E3F I          1E40 VAL      0E3F PRINTPHYSDEVICE      1E42 INDEX
1E43 I          0E71 PRINTBAUDRATE      1E44 INDEX      1E45 K
1E46 BAUD      0EC6 PRINTPHYSCHARACTERISTICS      1E47 INDEX      1E48 CHAR
1E49 CT          0F53 NAMES      1E4A I          1E4B J          1E4C COLS
1E4D CHAR      1E4E BAUD      1E4F K          0F85 CRLOOP      0F8A PROCESS
0FE0 SHOWPHYSICALDEVICES      1E50 VECTOR      1E52 DEVICEPRESENT
1E53 BITTABLE      1E63 I          1E64 K          1E65 COLS      1E66 MAX
1058 SETHAX      10F3 VALUES      1E67 VAL      116E SEARCHPHYSICALTABLE
1E69 SEARCHSTRINGADR      1E6B I          1E6C J          1E6D STRING
1210 SEARCHNEXT      1223 PROCSOPTION      1E73 TABLEINDEX      1E74 SOFTBAUD
1E75 CHAR      1E76 BAUD      1E77 VAL      1349 NUMBER      1E79 LOC
1E7B LENGTH      1E7C VAL      1E7E I          13CA PRINTBYTE      1E7F NUM
1E80 HUNDREDS      1E81 TENS      1E82 ONES      1449 PROCESSPAGEOPTIONS
1E83 NUM      1553 SHOWASSIGNMENTS      1E84 INDEX      1E85 OFFSET
1E86 VAL      1673 FOUNDLOGICALDEVICE      1E88 SAVEINDEX      1E89 OFFSET
1E8A EDLN      1E8B I          1E8C VAL      1913 SHOWCHARACTERISTICS
1E8D INDEX      1E8E CHAR      1E8F BAUD      1E90 J          1E91 I
19F9 FOUNDPHYSICALDEVICE      1E92 STRINGADR      1E94 EDLN      1E95 INDEX
1A62 PARSER      1E96 T          1E97 CHAR      1E98 I          1E99 EDLN
1E9A PHYSDEV      1E9B LOGDEV      1B41 INPUTFOUND      1E9C BUFFERADR

```

COPYRIGHT ©1982

DIGITAL RESEARCH

P.O. BOX 579

PACIFIC GROVE, CA 93950

SER. ==

0000 DEVICE#
0000 HCD80A#
0000 DEVICE#

084B 44	084C 46	0857 47	0858 48	085E 50
0867 51	0868 52	0870 56	087A 57	0883 58
088A 59	088F 60	088F 61	088F 62	0898 63
0898 64	089C 66	08A2 67	08A7 68	08B0 69
08B0 70	08B4 72	08BA 73	08BF 74	08CB 75
08C8 76	08CE 79	08D4 80	08D9 81	08E1 82
08E9 83	08EA 84	08F2 87	08FB 88	08FD 89
0907 90	090F 91	0910 92	0914 94	091A 95
0923 96	0923 97	0927 99	092C 100	093B 101
093E 102	094E 103	0952 104	0956 105	0959 106
095A 107	0A80 110	0A80 112	0A86 113	0A91 114
0AA2 115	0AA6 116	0AAD 117	0ABE 118	0ABF 119
0ACD 120	0AD0 121	0AD5 122	0AD6 123	0AD6 124
0AD6 125	0ADD 126	0AE0 127	0AE6 128	0AEB 129
0AF3 130	0AF4 131	0AF4 132	0AF8 133	0B06 134
0B0E 135	0B1F 136	0B24 137	0B28 138	0B33 139
0B34 140	0B34 141	0B42 142	0B4C 143	0B61 144
0B65 145	0B70 146	0B7A 147	0B7D 148	0B83 149
0B8F 150	0B90 151	0B90 152	0B90 154	0BA1 155
0BA8 156	0BAB 157	0B6B 158	0B6E 159	0973 160
0976 161	097F 162	0986 163	0990 164	0995 165
099D 166	09A0 167	09A7 168	09AA 169	09B4 170
09B7 171	09BA 172	09BD 173	09C0 174	09C6 175
09C9 176	09CA 177	09D0 178	09D3 179	09DA 180
09DF 181	09E6 182	09E9 183	09F2 184	09F5 185
0A11 186	0A19 188	0A1E 189	0A2D 190	0A30 191
0A37 192	0A3A 193	0A41 194	0A44 195	0A47 196
0A4C 197	0A4F 198	0A50 199	0A57 200	0A65 201
0A68 202	0A70 203	0A7A 204	0A7F 205	0A80 206
0BAC 207	0BB0 209	0BB8 210	0BC0 211	0BC6 212
0BCA 213	0BCA 214	0BCA 215	0BCF 216	0BD4 217
0BD5 218	0BE2 220	0BEE 221	0BF5 222	0BFC 223
0BFF 224	0C00 225	0D53 231	0D57 233	0D66 234
0D6D 235	0D74 236	0D7B 237	0D7E 238	0D8D 239
0D9A 241	0D9F 242	0DA4 243	0DA7 244	0DAC 245
0DB3 246	0DBA 247	0DBD 248	0C04 249	0C09 250
0C12 251	0C1A 252	0C1F 253	0C31 254	0C3F 255
0C57 256	0C61 257	0C70 258	0C73 259	0C82 260
0C8B 261	0C8F 262	0C96 263	0C9D 264	0CA4 265
0CAD 266	0CB3 267	0CBB 268	0CBE 269	0CCE 270
0CD7 271	0CE0 272	0CE9 273	0CF2 274	0CFB 275
0D04 276	0D0D 277	0D16 278	0D1F 279	0D28 280
0D31 281	0D47 282	0D4A 283	0D52 284	0DBE 285
0DC6 289	0DCC 290	0DD3 291	0DD8 292	0DE7 293
0DF5 294	0DF7 295	0DF7 296	0DFB 299	0E01 300
0E0E 301	0E1D 302	0E2C 303	0E33 304	0E3E 305
0E3F 306	0E43 308	0E51 309	0E69 310	0E70 311
0E71 312	0E75 314	0E89 315	0E90 316	0E95 317
0EA3 318	0EBE 319	0EC5 320	0EC6 321	0ECA 323
0ECF 324	0EE3 325	0EE8 326	0EEE 328	0EF3 329
0EF7 330	0EF7 331	0EFF 332	0F05 334	0F0A 335
0F0E 336	0F0E 337	0F18 338	0F1E 340	0F23 341
0F27 342	0F27 343	0F2F 344	0F35 346	0F3A 347
0F3E 348	0F3E 349	0F46 350	0F4B 351	0F4F 352
0F52 353	0F53 354	0F53 356	0F56 357	0F5C 358
0F62 359	0F68 360	0F6D 361	0F72 362	0F82 363
0F85 364	0F8A 365	0F9C 367	0F9F 368	0FA0 369
0FA0 370	0FA7 371	0FAC 372	0FB3 373	0FB8 374
0FBF 375	0FC5 376	0FC9 377	0FD2 379	0FD5 380
0FD8 381	0FD8 382	0FDC 383	0FDF 384	0FE0 385

COPYRIGHT ©1982
DIGITAL RESEARCH
P.O. BOX 579
PACIFIC GROVE, CA 93950

SER. # _____

0FE6 390	0FEC 391	OFF1 392	OFF6 393	1008 394
1016 395	101D 396	1028 397	102F 398	1034 399
1046 400	104E 401	1051 402	1055 403	1058 404
105E 405	1063 406	1068 407	1074 408	1083 410
108D 412	1093 413	1099 414	109C 415	109D 416
109D 417	10A2 418	10A9 419	10AE 420	10B2 421
10B9 423	10BE 424	10C1 425	10C7 426	10C7 427
10C7 428	10CE 429	10D6 431	10DB 432	10E1 433
10E1 434	10E9 435	10EF 436	10F2 437	10F3 438
10F3 440	10F6 441	10FC 442	1104 443	110A 444
1112 445	111A 446	1120 447	1128 448	1130 449
1136 450	113E 451	1146 452	114C 453	1154 454
115C 455	1162 456	116A 457	116D 458	116E 459
1174 464	117D 465	1180 466	118B 467	119B 468
11B2 469	11B9 470	11BE 471	11D0 472	11DE 473
1202 474	1205 475	120C 476	1210 477	1214 478
121A 479	121D 480	1220 481	1223 482	1223 483
1227 488	123C 489	1249 490	124E 491	1253 492
126B 493	1278 494	1280 495	1285 496	128D 497
1292 498	129A 499	12B0 500	12B8 501	12CE 502
12D7 504	12DF 505	12EA 506	12EE 507	12F3 508
12F8 509	130C 510	131F 511	1327 512	132F 513
1332 514	1345 515	1348 516	1349 517	1351 521
135A 522	135F 523	1365 524	1374 525	138A 526
138F 527	13A4 528	13AB 529	13B5 530	13C0 531
13C5 532	13CA 533	13CA 534	13CE 536	13DE 537
13F2 538	13FF 539	1410 540	1419 541	1422 542
1436 543	143F 544	1448 545	1449 546	1449 548
144E 549	1453 550	146B 551	1478 552	1480 553
1485 554	148D 556	1495 557	149D 559	14AA 560
14C1 561	14CA 562	14CA 563	14CA 564	14D2 566
14DA 567	14E2 569	14EF 570	1506 571	150F 572
150F 573	150F 574	1512 575	151A 576	1522 577
1525 578	152B 579	1533 580	1539 581	1541 582
1547 583	154A 584	1552 585	1553 586	1557 589
1567 590	157F 592	1585 593	158D 594	1595 595
159B 596	15A3 597	15AB 598	15AB 599	15C3 601
15C9 602	15D1 603	15D9 604	15DF 605	15E7 606
15EF 607	15EF 608	15F8 610	160A 611	1613 612
1616 613	161F 614	1622 615	162B 616	162E 617
1637 618	163A 619	1643 620	1655 621	165F 622
1667 623	1667 624	166A 625	1672 626	1673 627
1673 630	1679 631	1691 632	169B 634	16A3 636
16D3 637	16D9 638	16DE 639	16E1 640	16E9 641
16EE 642	16EE 643	16F3 644	16F8 645	1705 646
1715 647	171D 649	1725 651	172A 652	172D 654
1735 656	173A 657	173D 659	1747 660	1747 661
1747 662	174A 664	174F 665	175B 666	1766 667
1770 668	177B 669	178A 670	1792 671	1797 672
17A2 673	17AA 674	17B1 675	17E2 676	17E7 677
1800 678	1807 679	180F 680	181C 681	181F 682
1837 684	184F 686	1854 687	1859 688	185C 689
1861 690	1864 692	187C 694	1894 696	1899 697
189E 698	18A1 699	18A6 700	18A9 702	18C1 704
18D9 705	18E1 706	18EB 707	18EB 709	1903 710
190B 711	1912 712	1912 713	1912 714	1912 715
1912 716	1913 717	1917 719	192B 720	193F 721
1942 722	1948 723	194F 724	1952 725	1958 726
195F 727	1962 728	196B 729	1976 730	197E 731
1984 733	1995 734	1998 735	19A6 736	19AB 737
19B5 738	19B8 740	19C0 742	19C6 743	19C9 744
19D7 745	19DC 746	19E6 747	19E6 748	19E6 749
19F0 750	19F8 751	19F9 752	19FF 755	1A17 756
1A1F 757	1A24 758	1A2F 759	1A37 760	1A3C 761
1A43 762	1A4A 763	1A52 764	1A5C 765	1A61 766
1A62 767	1A62 768	1A67 769	1A6C 771	1A71 775

COPYRIGHT ©1982
DIGITAL RESEARCH
P.O. BOX 579
PACIFIC GROVE, CA 93950

SER. # _____

1A80 777	1A86 778	1A8A 779	1A8F 780	1A8F 781
1A9C 782	1AC0 783	1AC8 784	1ACD 785	1AE5 787
1AF4 788	1AF7 789	1AFA 790	1AFD 792	1B05 794
1B0C 795	1B12 796	1B17 797	1B1A 799	1B22 801
1B29 802	1B2F 803	1B34 804	1B37 806	1B3A 807
1B3D 808	1B40 809	1B40 810	1B40 811	1B40 812
1B41 813	1B47 816	1B61 817	1B68 818	1B6B 819
1B74 820	1B77 821	1B7A 822	06EA 823	06F0 824
0708 826	070E 827	0716 828	0716 829	071E 830
073A 832	0740 833	074D 834	0771 836	0774 837
077C 838	077C 839	077C 840	0788 842	078E 843
0796 844	0796 845	079E 846	07A6 847	07BA 848
07C5 850	07C8 851	07CB 852	07D1 853	07D4 854
07DF 855	07E2 856	07E7 857	07F2 858	07FD 859
0805 860	0814 861	082B 862	0835 863	083B 864
0838 865	083E 866	0846 868		

COPYRIGHT © 1982
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950
 SER. # _____